

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 05-216845

(43)Date of publication of application : 27.08.1993

(51)Int.Cl. G06F 15/16
G06F 13/00

(21)Application number : 04-262099

(71)Applicant : INTERNATL BUSINESS MACH
CORP <IBM>

(22)Date of filing : 30.09.1992

(72)Inventor : ALLON DAVID
BACH MOSHE
MOATTI YOSEF
TEPERMAN ABRAHAM

(30)Priority

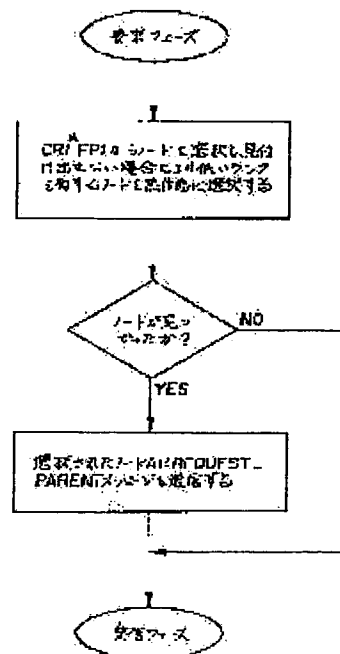
Priority number : 91 99923 Priority date : 31.10.1991 Priority country : IL

(54) METHOD FOR OPERATING COMPUTER IN NETWORK

(57)Abstract:

PURPOSE: To balance a load as quickly as possible.

CONSTITUTION: A computer forms a logical link with another computer in a network so that a tree structure can be formed, logically links with one computer in a tree higher order, and logically links with the computer in a tree lower order. Information related with the present load of the computer and the loads of at least several computers among the other computers is held in the computer, this information is periodically distributed by the computer in which the information is stored to the plural computers logically linked with the computer, the information of itself is updated according to the information, and the computer in the network which can receive the excessive load is judged.



(19)日本国特許庁(JP)

(12)公開特許公報(A)

(11)特許出願公開番号

特開平5-216845

(43)公開日 平成5年(1993)8月27日

(51)Int.Cl. ⁵	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 15/16	3 8 0 Z	9190-5L		
13/00	3 5 5	7368-5B		

審査請求 有 請求項の数9(全15頁)

(21)出願番号 特願平4-262099

(22)出願日 平成4年(1992)9月30日

(31)優先権主張番号 99923

(32)優先日 1991年10月31日

(33)優先権主張国 イスラエル(IL)

(71)出願人 390009531

インターナショナル・ビジネス・マシー
ズ・コーポレイション

INTERNATIONAL BUSIN
ESS MACHINES CORPO
RATION

アメリカ合衆国10504、ニューヨーク州
アーモンク (番地なし)

(72)発明者 ダビッド アロン

イスラエル エルサレム メイヤー ナカ
ル ストリート 49/8

(74)復代理人 弁理士 谷 義一 (外3名)

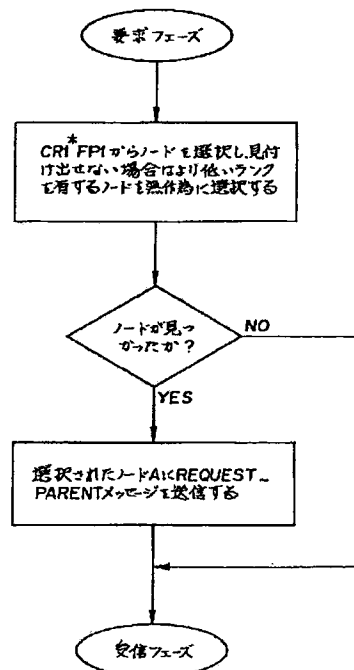
最終頁に続く

(54)【発明の名称】 ネットワーク内のコンピュータを操作する方法

(57)【要約】

【目的】 より速く負荷を平衡させる。

【構成】 ツリー構造が形成されるように、当該コンピ
ュータはネットワーク内の他のコンピュータと論理リン
クを形成し、ツリー上位の1つのコンピュータに論理的
にリンクし、そのツリー下位のコンピュータに論理的に
リンクする。当該コンピュータの現在負荷と、前記他の
コンピュータのうちの少なくとも幾つかのコンピュータ
の負荷とに関する情報をコンピュータ内に保持し、その
コンピュータに論理的にリンクされた複数のコンピ
ュータに、前記情報が記憶されたコンピュータが前記情報を
周期的に配送し、かつ、その情報に従ってそれ自体の情
報を更新し、その情報を用いて、余分の負荷を受け取る
ことができるネットワーク内のコンピュータを判定す
る。



【特許請求の範囲】

【請求項1】 ツリー構造が形成されるように、ネットワーク内の当該コンピュータと他のコンピュータの間に論理リンクを形成し、前記当該コンピュータはツリーの上位の1つのコンピュータに論理的にリンクされるとともに、そのツリーの下位のコンピュータに論理的にリンクされるステップと、
前記ネットワーク内の前記当該コンピュータの現在負荷と、前記他のコンピュータのうちの少なくとも幾つかのコンピュータの負荷とに関する情報であって、かつ記憶された情報を、前記コンピュータに論理的にリンクされた複数のコンピュータに周期的に配送し、前記複数のコンピュータから同様のこのような情報を受信し、かつ、その情報に従ってそれ自体の情報を更新することにより、前記情報を前記コンピュータに保持し、その結果、その情報を用いて、余分の負荷を受け取ることができるネットワーク内のコンピュータを判定するステップとを備えたことを特徴とするネットワーク内のコンピュータを操作する方法。

【請求項2】 前記コンピュータが過負荷になると、前記情報を用いて、余分の負荷を受け取ることができるコンピュータを判定し、そのコンピュータに少なくとも1つのタスクを伝送するステップを備えたことを特徴とする請求項1に記載のネットワーク内のコンピュータを操作する方法。

【請求項3】 前記コンピュータに論理的にリンクされているツリー上位のコンピュータに障害が発生するか、あるいは動作不能になった場合、新しい下方リンクを受け取る容量を有するネットワーク内の別のネットワークに新しく論理リンクを生成するステップを備えたことを特徴とする請求項1または2に記載のネットワーク内のコンピュータを操作する方法。

【請求項4】 前記コンピュータにより負荷情報を周期的に配送して、前記コンピュータにリンクされている前記ツリー内のコンピュータが動作可能か否かを判定し、前記負荷情報の周期的な配送が可能であり、かつ、前記コンピュータがリンクされているツリーの下位のコンピュータの1つが動作不可能な場合、そのコンピュータは新しい下方リンクを受け取る容量を有するものとして印を付けるステップを備えたことを特徴とする請求項1ないし3のいずれかに記載のネットワーク内のコンピュータを操作する方法。

【請求項5】 前記コンピュータに記憶された情報は多くの項目を含み、各項目は

(i) 前記ネットワーク内の前記コンピュータのうちの1つのコンピュータの負荷、

(i i) コンピュータとコンピュータを分離するツリー内のリンクの数、

(i i i) 前記コンピュータにリンクされ、前記項目が最後に受信される前記コンピュータのうちの1つのコン

ピュータとに関する情報を含み、かつ、

前記コンピュータがそのコンピュータにリンクされているコンピュータから、同様の情報を受信すると、

(a) 受信された情報の各項目内のリンクの数を1だけインクリメントするステップと、

(b) 前記コンピュータから発信され、かつ、受信された情報内の項目を削除するステップと、

(c) 送信するコンピュータから受信されたコンピュータ内に既に記憶されている情報の項目を削除するステップと、

(d) ローカルベクトルと受信されたベクトルの両方に現れる項目（コンピュータ）の距離を比較するステップと、

1. ローカル項目の距離が受信された項目の距離より大きい、あるいは等しい場合、ローカル項目を削除する。

2. 受信された項目の距離がローカル項目の距離より大きい、あるいは等しい場合、その受信された項目を削除する。

(e) その受信された項目を既に前記コンピュータに記憶されている情報と併合するステップと、

(f) 併合された情報を負荷の昇順に分類し、等しい負荷を有する項目を、前記コンピュータからのリンク分離数の昇順で分類するステップとを実行することを特徴とする請求項1ないし4のいずれかに記載のネットワーク内のコンピュータを操作する方法。

【請求項6】 情報をそれぞれ配送する前に、同一の負荷を有するコンピュータに関する情報の項目と、前記コンピュータからのリンク分離数とを無作為に置換することを特徴とする請求項5に記載のネットワーク内のコンピュータを操作する方法。

【請求項7】 前記コンピュータに対応する項目が最初に前記コンピュータにより分類された情報に現れた場合、カウンタが前記項目に接続され、かつ、前記コンピュータの負荷に初期設定され、その項目が依然最初であり、情報が配送された場合はいつでも、前記カウンタはインクリメントされ、前記カウンタが正になった場合、その項目は前記情報から取り外し、そのことにより、異なるコンピュータに記憶された情報に同一の項目が最初に現れる確率を小さくすることを特徴とする請求項5または6に記載のネットワーク内のコンピュータを操作する方法。

【請求項8】 同様のこのようなコンピュータのネットワークで操作することができるコンピュータであって、コンピュータがツリー構造で論理的にリンクされたネットワーク内のコンピュータを識別する識別手段と、前記コンピュータ上の負荷と前記ネットワーク内の他のコンピュータの少なくとも幾つかに関する情報を記憶する記憶手段と、

前記コンピュータが論理的にリンクされたコンピュータ

に前記情報を送信する送信手段と、
前記コンピュータが論理的にリンクされているコンピュータから同様の情報を受信し、かつ、前記記憶された情報を、受信された情報に従って更新する更新手段と、
余分の負荷容量を有するネットワーク内の他のコンピュータの1つを前記情報から選択し、かつ、選択されたコンピュータにタスクを転送する転送手段とを備えたことを特徴とするコンピュータ。

【請求項9】 前記ネットワーク内の全てのコンピュータのリストを記憶するか、あるいはアクセスする手段と、

前記リストから代表の隣接コンピュータとして1つのコンピュータを選択し、前記ツリー構造にリンクする手段と、

前記選択されたコンピュータにメッセージを送信し、リンク要求を示す手段と、

前記選択されたコンピュータから肯定応答が受信された場合、前記識別手段を更新することにより、前記選択されたコンピュータとの論理的なリンクを確立する確立手段と、

他のコンピュータからのメッセージであって、リンクが要求されていることを示すメッセージを受信する受信手段とこのようなリンク要求メッセージを受信したとき、
下方リンクを確立するだけの容量があるか否かを判定し、このような容量が存在する場合、前記リンク要求メッセージの送信元に肯定応答を送信し、これに従って、前記識別手段を更新する手段とを備えたことを特徴とする請求項8に記載のコンピュータ。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、ネットワークのコンピュータを動作させる方法に関し、かつ、ネットワークで動作させることができるコンピュータに関する。

【0002】

【従来の技術】コンピュータのLAN (local area network) は、多くの組織では欠くことのできないものである。コンピュータの大きなネットワークの問題の1つとして、ネットワークのコンピュータを全て効率的に利用するには困難なことがあるということである。負荷分割すなわち平衡方法では、全てのコンピュータをビジー状態にしようとするにより、システムスループットがますます高められる。この平衡方法は過負荷状態のコンピュータからプロセスをオフロードしてコンピュータをアイドル状態にすることにより実施され、よって、全ての計算機上の負荷が等化され、全体的な応答時間を最小にする。

【0003】負荷平衡法は平衡を達成するために用いられる方法に従って分類される。それらは、「静的」、「動的」、「集中型」、および「分散型」に分類される。

【0004】静的負荷平衡法では、決定的または確率的にかかわらず、現行のシステムの状態とは独立に固定的な方針に従う。静的負荷平衡法はインプリメントするには簡単であり、かつ、待ち行列モデルを用いて分析するには容易である。しかし、その潜在的な利点は限定されている。というのは、静的負荷平衡法では大域的なシステム状態の変化を考慮に入れていないからである。例えば、コンピュータは既に過負荷状態にあるコンピュータにタスクを移行させることができる。

【0005】動的負荷平衡法では、そのシステムは大域的なステータスの変化に注意し、コンピュータの現在の状態に基づいてタスクを移行させるか否かを判定する。動的負荷平衡法は、静的負荷平衡法より本来複雑である。というのは、動的負荷平衡法は、システム内の他のコンピュータの状態を知るコンピュータを必要とするからである。他のコンピュータの状態の情報は絶えず更新しなければならない。

【0006】集中型負荷平衡法では、1つのコンピュータは大域的な状態情報を含み、全ての判定を行う。分散型負荷平衡法では、大域的な情報を有するコンピュータは1台もない。各コンピュータは、当該コンピュータが有する状態情報に基づき、移行するか否かの判定を行う。

【0007】負荷平衡法はさらに「送信元開始型」および「受信先開始型」に分類される。送信元開始型方策では、過負荷状態のコンピュータは負荷の軽いコンピュータをシークする。受信先開始型方策では、負荷が軽いコンピュータがタスクを受信する能力を広告 (advertise) する。タスク転送コストが送信元開始型および受信先開始型のもで同等である場合、送信元開始型方策はシステム負荷を加減するため、受信先開始型方策をアウトパフォームする。

【0008】通常、動的負荷平衡機構は次の3つのフェーズよりなる。

【0009】1. ローカル計算機の負荷を測定する。

【0010】2. ローカル負荷情報をネットワークの他の計算機と交換する。

【0011】3. 選択された計算機にプロセスを転送する。

【0012】ローカル負荷は、実行待ち行列、記憶装置利用率、ページングレート、ファイル使用、および入出力レート、または他の資源利用率のジョブの数の関数として計算することができる。実行待ち行列の長さは通常受信される負荷メトリックである。

【0013】他のノードから負荷情報を受信することによりシステム負荷のスナップショットが与えられる。この情報を得ると、各コンピュータは転送方針を実行し、タスクを局所的に実行するか、あるいはタスクを別のノードに転送するかを判定する。判定した結果、タスクを転送する場合は、ロケーション方針が実行され、タスク

を転送すべきノードを決定する。

【0014】多くのコンピュータを有するネットワークで、負荷平衡法が効率的である場合、負荷平衡法は、ネットワークトラフィックがネットワークのサイズに対して線形的に増加するという意味で拡張可能でなければならないし、コンピュータをネットワークに付加したりネットワークから除去したりすることが容易にできるように柔軟性がなければならないし、1つ以上のコンピュータに障害が生じた場合頑強でなければならないし、コンピュータのクラスタ化をある程度サポートできなければならない。

【0015】集中化方法は、簡単であるので魅力的ではあるが、フォールトトレラントな方法ではない。中央コンピュータは死んだノードを検知することができ、かつ、1つまたは2つのメッセージでコネクションを再確立する。しかし、中央ノードに障害が生じた場合は、その方式全体がだめになる。各ノードで優先権を与えられた別の管理者のリストを保持するか、あるいは選挙法をインプリメントして新しく中央ノードになるノードを選挙することにより、負荷平衡を再構成することができる。中央ノードに障害が生じると、他のノードは新しい中央ノードに切り換える。しかし、このようにすると、管理がますます複雑になり、しかも、その方法のコストが高くなる。

【0016】さらに、中央化方法に対する管理オーバーヘッドは、大きすぎて受託することができない。ノードの数が増加するにつれて、中央のノードが負荷情報を処理するのに費やす時間が増加し、ついには、中央ノードが過負荷状態になるに違いない。従来の負荷平衡方法には動的および分散型であるものが幾つかある。

【0017】例えば、Barak A. and Shilo A. SOFTWARE-PRACTICE AND EXPERIENCE, 15 (9) : 901-913, September 1985 [R1]に記載された方法では、固定されたサイズの負荷ベクトルであって、かつ、コンピュータ間に周期的に配送される負荷ベクトルを、各コンピュータが保持している。各コンピュータは次の方法を実施する。まず、コンピュータはそのコンピュータ自体の負荷値を更新する。ついで、そのコンピュータは無作為に1つのコンピュータを選択し、選択されたコンピュータに、そのコンピュータの負荷ベクトルの半分の一方を送信する。負荷ベクトルを受信すると、各コンピュータは、予め規定された規則に従って、情報をローカル負荷ベクトルと併合する。

【0018】

【発明が解決しようとする課題】この方法の問題点は、負荷ベクトルのサイズを注意深く選択しなければならないことである。負荷ベクトルに対する適正な値を見つ出すことは困難であり、任意のシステムに適合しなければならない。大きい負荷ベクトルは多くのノードに対す

る負荷情報を有する。従って、ノードは負荷の軽いノードを知って、移行する多くのプロセスを直ちに受信するという機会が増加し、過負荷になるものが多い。一方、負荷ベクトルのサイズは適正な時間でネットワークを伝播できないほど小さくすべきでない。ネットワークのn個のノードに対して、負荷情報をネットワークを介して伝播する予期した時間は $O(\log n)$ である。

【0019】R1に記載された方法は、負荷ベクトルの長さを調整した後任意の数のコンピュータを処理することができる。そのため、その方法は拡張が可能である。しかし、柔軟性のみがある程度ある。小数のノードをネットワークに付加するには、全てのコンピュータの負荷ベクトルのサイズを変更する必要がある。従って、管理オーバーヘッドを増加する。その方法はフォールトトレラントがあり、単一の障害にも動作し続ける能力がある。しかし、デッドノードを検知し、かつ、負荷ベクトル内のデッドノードに関する情報を更新するための組み込み機構がない。従って、障害が生じたノードの情報は伝播されないが、他のノードはプロセスを障害が生じたノードに移行し続けることができる。その結果、応答時間が減少する。

【0020】単位時間当たりの通信数は $O(n)$ である。しかし、その方法はクラスタ化をサポートしない。1つのノードか他のノードにプロセスをオフロードすると、そのノードはそのノードの負荷ベクトルから対象を選択する。ノードpの負荷ベクトル内のノードに関する情報が無作為のノードから更新されるので、オフロードする対象の集合は無作為のnの部分集合であって、制御されたpの集合ではない。

【0021】Lin F. C. H. and Keller R. M. PROCEEDINGS OF THE 6TH INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS, 329-336, May 1986 [R2]には、傾斜モデルを用いた分散型および同期負荷平衡方法が開示されている。コンピュータがグリッドに論理的に配置され、コンピュータの負荷が表面として表されている。

【0022】各コンピュータはグリッド上の隣接するコンピュータとのみ対話する。負荷の大きいコンピュータは「丘」であり、アイドル状態のコンピュータは「谷」である。中性コンピュータは過負荷状態でもなくアイドル状態でもないコンピュータである。負荷平衡はその表面の弛張の一形態である。タスクは丘から谷に移行し、表面が平らになる。各コンピュータは最も近いアイドル状態のコンピュータまでの距離を計算し、その距離を用いてタスクを移行させる。

【0023】過負荷状態のノードはタスクをアイドル状態のノードの方向にある近接するコンピュータに転送する。タスクはアイドル状態のノードに到達するまで移動

し続ける。タスクが到達したとき、最初のアイドル状態のノードが過負荷状態にされた場合、そのタスクは別のアイドル状態のノードに移動する。

【0024】この方法は拡張可能であるが、部分的にのみ柔軟性がある。グリッドの縁にノードを付加したり、グリッドの縁のノードを取り除いたりすることは容易であるが、グリッドの真中にノードを付加したり、その真中のノードを取り除いたりするのは困難である。というのは、そのグリッドが固定されているからである。ノードを付加したり取り除いたりすると、再構成が必要である。その再構成によりその方法が備えているオーバーヘッドが増加する。死んだノードを検知することは容易ではない。ノードの状態の変化のみが近接するコンピュータに送信されるので、ノードの障害は、ジョブが近接するコンピュータに転送されるまで、検知されないままである。検知が遅れるとプロセスの移行が遅られ、そのため、全体的な応答時間が増加する。この方法では、クラスタ化はサポートされていない。異なる方向に転送することができる近接するコンピュータのうちの1つに、各ノードはプロセスを転送する。過負荷状態のノードは、オフロードされたプロセスを結果的にどこで実行するかは制御しない。そのグリッドを開始するための管理オーバーヘッドは小さく、メッセージの数は $O(n)$ である。再構成に対する管理のオーバーヘッドは大きい。というのは、障害の生じたノードに対して近接するノードが変化が必要であるからである。この方法では、ノードオーバーヘッドは大きく、ネットワークのトラフィックが増加する。というのは、タスクはアイドル状態のノードに直接移動するというよりは、ホップして移動し、しかも、プロセスの移行に時間がかかるからである。

【0025】Shin K. G. and Chang Y. C. IEEE TRANSACTION ON COMPUTERS, 38 (8), August 1989 [R3] にて提案されている仲間集合 (buddy set) アルゴリズムは、負荷分割を非常に速く行うことを目的としている。

【0026】各ノードに対して2つのリストが規定されている。仲間集合、すなわち、その近接するコンピュータの集合と、その好ましいリスト、すなわち、タスクが転送される仲間集合内の順序付けされたノードのリストである。各ノードは次の状態のうちの1つの状態になることができる。すなわち、標準より小さい負荷がかかった状態 (under-loaded)、中ぐらいの負荷がかかった状態 (medium-loaded)、過負荷状態である。各ノードは仲間集合内の全てのノードのステータスを含む。ノードのステータスが変化した場合はいつでも、ノードは新しい状態をその仲間集合の全てのノードにブロードキャストする。好ましいリストの内部順序は、仲間集合内の2つのノードが同一のノードにタスクを転送する確率を最小にするためノードごとに変

更する。1つのノードが過負荷状態になると、そのノードはその好ましいリストを走査して標準より小さい負荷がかかったノードを探し、そのノードにタスクを転送する。過負荷状態のノードは好ましいリストから外される。仲間集合およびそれらの好ましいリストはオーバーラップしており、そして、プロセスはそのネットワークを巡回する。

【0027】再度述べるが、この方法は、メッセージの数がネットワーク n のコンピュータの数と線形的に増加する点では拡張可能であるが、柔軟性がない。ネットワークにノードを付加したり、ネットワークからノードを取り除くには、関係する仲間集合の全てのノード内の好ましいリストを再計算する必要がある。1つのノードか2つ以上の仲間集合に付加された場合は、各仲間集合の各ノードに対して再計算しなければならない。R2に記載された傾斜モデルと同様の理由で死んだノードを検知することは困難である。クラスタ化はサポートされているが、再構成はコストがかかる。というのは、ノードを付加したり取り除いたりするため、ノードは新しい仲間集合と好ましいリストを再計算する必要があるからである。そのため、その方法が備えた管理オーバーヘッドは大きい。

【0028】

【課題を解決するための手段】本発明に係る第1の実施態様では、コンピュータネットワークのコンピュータを走査する方法を提供する。その方法は、そのコンピュータとネットワーク上の他のコンピュータとがツリー構造を形成するように論理リンクを生成し、そのコンピュータはそのツリーの上位の1つのコンピュータと下位の多くのコンピュータとに論理的にリンクされ、前記ネットワーク内の前記当該コンピュータの現在負荷と、前記他のコンピュータのうちの少なくとも幾つかのコンピュータの負荷とに関する情報であって、かつ記憶された情報を、前記コンピュータに論理的にリンクされた複数のコンピュータに周期的に配送し、前記複数のコンピュータから同様のこのような情報を受信し、かつ、その情報に従ってそれ自体の情報を更新することにより、前記情報を前記コンピュータに保持し、その結果、その情報を用いて、余分の負荷を受け取ることができるネットワーク内のコンピュータを判定することができる。

【0029】本発明は送信元開始型および受信先開始型負荷平衡法に適用可能である。送信先開始型負荷平衡方式を用いた本発明の実施例は、コンピュータが過負荷状態になると、その情報を用いて、余分の負荷を受け取ることができるコンピュータを判定し、そのコンピュータに少なくとも1つのタスクを転送するステップをさらに含む。

【0030】上述する方法を用いて、各コンピュータを操作することにより、コンピュータネットワークを操作する方法をさらに提供する。

【0031】ランクを各コンピュータに割り当てることにより、論理リンクを生成することができる。どの2つのコンピュータにも同一のランクが割り当てられない、各コンピュータはそれより下位のランクの1つのコンピュータと、それより上位のランクの多くのコンピュータに論理的にリンクし、ツリーを形成する。

【0032】コンピュータに障害が生じるか、あるいは操作不能になった場合、障害が発生したコンピュータがリンクされているツリーの下位の各コンピュータと、新しい下位のリンクを受け取る容量を有する他のコンピュータとの間に新たに論理リンクを生成することにより、ツリー構造を保守することができる。

【0033】改善された負荷平衡方式が採用され、この方式の特徴は、各コンピュータの通信負荷と、各コンピュータに記憶するのに必要なネットワークブローギの知識を制限することにより拡張するという点にある。従って、各コンピュータのアクションはネットワーク上のコンピュータの総数とは無関係である。

【0034】ツリー構造は動的に構築されかつ保守されるので、その方法はまた柔軟性がある。ツリー構造を変化させるには、再構成されたコンピュータのランク付けのみを変化させる必要がある。新しいランク付けファイルは、循環することを避けるため、現在のランク付けファイルと一致していなければならない。そのコンピュータが隣接するコンピュータから見て操作不能に見えるように、そのコンピュータを論理的に切断し、その構成ファイルを交換し、かつ、コンピュータをツリーと再コネクションさせることにより、コンピュータが再構成される。ツリーの他のノードを乱したり、ローカルサービスを遮断せずに新たに構成される。

【0035】また、障害が生じたノードを検知し、かつ、そのノードを再コネクションすることが提供されている。ツリーの上位の1つのコンピュータがある時間内で応答しない場合は、その1つのコンピュータと前もってコネクションされている上位のランクのコンピュータは、そのツリーのどこかで再リンクを試みる。各上位のランクのコンピュータが再リンクに一時的に失敗した場合、各上位のランクのコンピュータは切断されたサブツリーのルートとして動作し、そのサブツリー内で負荷が平衡する。結果的に、既に試みたノードがあるいは死んだノードをマークするフラグはリセットされることになり、ツリーに周期的に接続を試みるコンピュータはそのツリーと再コネクションすることになる。

【0036】都合のよいことに、負荷情報が各コンピュータによりツリーのリンクを介して配送され、各コンピュータにリンクされたコンピュータが操作可能か否かを判定する。各コンピュータとツリーの上位でリンクされているコンピュータが操作不可能である場合、各コンピュータはシークし、下位のランクのコンピュータと新たにリンクを生成するとともに、新しい下位のリンクと

リンクする能力を有するコンピュータの1つと新たにリンクを生成する。各コンピュータがリンクされたツリーの下位のコンピュータの1つが、新しい下位のリンクを受け取る容量を有するものの、操作不可能である場合、各コンピュータはマークされる。

【0037】本発明の好ましい形態では、各コンピュータに記憶された情報は多くの項目を含む。各項目はネットワーク内のコンピュータの特定の1つの負荷に関する情報と、その情報が記憶されたコンピュータから*i*個に分離されたツリーのリンクの数*i*と、前記負荷情報が記憶されたコンピュータに論理的にリンクされたコンピュータの名前（すなわち、ランク）を含む。負荷情報が記憶されたコンピュータから前記各項目が最後に受信される。各コンピュータに論理的にリンクされたコンピュータから、前記各コンピュータが同様の情報を受信すると、次のステップが実施される。

【0038】a) 受信された情報の各項目のリンク分離値の数は1だけインクリメントされる。

【0039】b) 受信された情報であって、かつ、情報を受信するコンピュータから発信された情報の項目は削除される。

【0040】c) 情報を受信するコンピュータに既に記憶された情報であって、かつ、情報を送信するコンピュータから受信された情報の項目は削除される。

【0041】d) ローカルベクトルおよび受信されたベクトルとして現れる項目（コンピュータ）のリンク分離数は比較される。

【0042】1. ローカル項目の数が大きいあるいは等しい場合、そのローカル項目は削除される。

【0043】2. 受信された項目の値が大きい場合、その受信された項目は削除される。

【0044】e) 受信された項目はその項目を受信するコンピュータに既に記憶された情報と併合される。

【0045】f) その併合された項目は負荷の昇順で記憶され、同等の負荷を有する項目は、項目を受信するコンピュータからのリンク分離数の昇順で分類される。

【0046】情報をコンピュータにそれぞれ配送する前に、同一の負荷と、コンピュータからのリンク分離の数とを有するコンピュータに関する情報の項目を無作為に置換することができ、その結果、異なるコンピュータに記憶された情報に同一の項目が最初に現れる確率が減少する。また、そのコンピュータに対応する項目がそのコンピュータの分類された情報に最初に現れる場合、その項目にカウンタを接続し、かつ、初期設定してそのコンピュータの余分な負荷容量を負の値にすることができる。その項目が依然として最初である場合、その情報が配送されるときはいつでも、そのカウンタがインクリメントされる。そのカウンタが正になると、その情報からその項目が除去される。よって、異なるコンピュータに記憶された情報に同一の項目が最初に現れる確率が減少

する。

【0047】本発明の好都合の形態では、その情報をツリーの上位にあるコンピュータに周期的に配送する期間は、そのコンピュータで作成される新しいタスクのレートおよび／またはツリー構造に論理的にリンクされたコンピュータで作成される新しいタスクのレートに関係する。1つ以上のノードの負荷が突然大きくなった場合、この情報がネットワークにより速く広がることになるという利点を有する。

【0048】さらに、本発明で提供される方法の利点は、ネットワークの全体的な負荷が絶えず増加している場合は礼儀正しく処理されるという点にある。このような場合、目標コンピュータに対して、ローカル負荷ベクトルが探索されると、候補がより少なくなり、移行数が減少することになる。負荷が全体的に減少すると、ローカル負荷ベクトルの候補がより多くなり、移行が再開される。言い替えると、ネットワーク性能はネットワーク負荷が飽和している間礼儀正しく低下し、クラッシュすることはない。むしろ、全体的な負荷が正常に戻ると、正常な動作が再開される。

【0049】別の態様から見ると、本発明は、同様のこのようなコンピュータのネットワークで動作することができるコンピュータを提供することができる。このコンピュータは、ツリー構造の中でコンピュータが論理的にリンクされるネットワークのコンピュータを識別する手段と、そのコンピュータおよびネットワーク上の少なくとも幾つかの他のコンピュータの負荷に関する情報を記憶する手段と、ツリー構造でそのコンピュータが論理的にリンクされているコンピュータに前記情報を送信する手段と、そのコンピュータが論理的にリンクされている前記コンピュータからの同様の情報を受信し、かつ、記憶された情報をその情報に従って更新する手段と、余分の負荷容量を有するネットワークの他のコンピュータの1つを前記情報を用いて選択し、かつ、選択されたコンピュータにタスクを転送する手段とを備えていることを特徴とする。

【0050】また、そのコンピュータは、ツリーにリンクされる隣接する候補のコンピュータとして1つのコンピュータをこのリストから選択する手段と、選択されたコンピュータにリンク要求を示すメッセージを送信する手段と、選択されたコンピュータから肯定応答が受信された場合、前記識別する手段を更新することにより、選択されたコンピュータとの論理的なリンクを確立する手段と、リンクが要求されていることを示す他のコンピュータからメッセージを受信する手段と、このようなリンク要求メッセージを受信したとき、ツリーの下位でリンクを確立する容量があるか否かを判定し、かつ、このような容量がある場合は肯定応答をリンク要求メッセージの送信元に送信し、したがって、前記識別する手段を更新する手段とを含むことが好ましい。

【0051】ツリーの下位のコンピュータからの同様の情報を当該コンピュータが受信すると、この受信にตอบสนองして、前記情報がツリーの下位のコンピュータにより配送される。

【0052】ランクを各コンピュータに割り当てることにより、論理リンクが生成され、かつ、任意の2つのコンピュータも同一ランクが割り当てられず、各コンピュータはより下位の1つのコンピュータにリンクされるとともに、より上位の多くのコンピュータにリンクされ、ツリー構造を形成する。

【0053】さらに、このようなコンピュータのネットワークを提供する。

【0054】

【実施例】本発明の実施例を図面を参照して説明する。

【0055】本発明に係る実施例は2つの主要部よりなる。第1の部分はツリー構築と保守である。第2の部分は負荷平衡情報の交換と保守である。

【0056】ツリー構築と保守

ネットワーク上の各コンピュータにはツリー構築に用いる一意のランクが割り当てられる。以下、「親」という語は下位ランクのコンピュータを参照するために用いられ、この下位のコンピュータに対して1つのコンピュータは上位のリンクを形成することができる。「子」という語は上位ランクのコンピュータをいい、この上位ランクのコンピュータに対して1つのコンピュータは下位のリンクを形成することができる。

【0057】各コンピュータは次の(i)～(v)を含む構成ファイルを記憶し、その構成ファイルにアクセスする。

【0058】(i) そのコンピュータが受け取ることができる直系の子孫の最大数。これらの数は子スロットという。

【0059】(ii) コンピュータのリストおよびコンピュータのそれぞれのランク。

【0060】(iii) 「好意的な親」ノードの順序付けリストFP。好意的な親のランクは現在コンピュータより下位である。FPリストはコンピュータにより用いられ親ノードを選択する。FPリストは任意に選択される。親が無作為に選択される場合にはFPリストを空にすることができる。FPリストは下位の全てのコンピュータの部分集合を含む。

【0061】(iv) 親および子の応答を待つ時間。

【0062】(v) ノードが死んでいるかあるいは生きているかをブローブするために待機する時間。

【0063】コンピュータCRに対する候補の親のリストは、Pの候補範囲というが、可能な親の範囲を限定する。はじめ、各コンピュータのCRはそのコンピュータより低いランクのコンピュータを全て含む。ノードがCRおよびFPにある場合にのみ、そのノードが親として選択される。FPが空であるか、あるいは使い果たした

場合は、CRのメンバーのみが検査される。

【0064】各コンピュータはツリーを構築するため、次のステップを実施する。要求フェーズがあり、このフェーズでは、1つのコンピュータは別のコンピュータを選択し、親になることを要求する。また、受信フェーズがあり、このフェーズでは、コンピュータは応答を待ち、他のコンピュータからの要求に応答する。

【0065】要求フェーズ

1. 各コンピュータ i は好意的な親のリスト FP_i から別のコンピュータを取り出す。 FP_i が空か、あるいは、 FP_i の全ての候補が失敗した場合、それより下位のコンピュータが無作為に選択される。選択されたコンピュータ j が CR_i にない場合は、 FP_i の次のコンピュータが選択されるか、あるいは、別のコンピュータが無作為に取り出される。全てのノードの試みが失敗した場合は、そのコンピュータは受信フェーズに入る。

【0066】2. i は選択されたコンピュータ j に $request_parent$ メッセージを送信し、かつ、他のコンピュータからのメッセージを待つ。

【0067】3. 代表の親が、ある期間内に応答しない場合、この代表の親は死んだものと印を付けられ、 i は要求フェーズに再び入る。

【0068】要求フェーズ

1. $request_parent$ メッセージが m から到達すると、(a) i が自由子スロットを有する場合、 i は m の要求を受信し、 ack メッセージを送信する。

【0069】(b) i が割り当てられた数の子スロットを使い果たした場合、

1) i は子を走査して次に規定する前の子を探す。前の子を見つけた場合、

(a) その子は死んだものと印を付けられ削除される。

【0070】(b) 子スロットを利用可能にする。

【0071】(c) i は m に ack メッセージを送信する。

【0072】前の子が見つからない場合は、

2) i は最も高いランクを有する子 k を見つける。

【0073】3) $rank(k) > rank(m)$ である場合、

a) i は k のかわりに m を子として受け入れ、 m に ack メッセージを送信する。

【0074】b) i は k に $rank(m)$ をパラメタとして含む $disengage$ メッセージを送信する。

【0075】4) $rank(k) < rank(m)$ である場合、

a) i はコンピュータの中のリストに $rank(k) > rank(m)$ を満足する最も低いランクを有するコンピュータ n を置く。

【0076】b) i は m に $rank(n)$ をパラメタとして含む $noack$ メッセージを送信する。

【0077】2. パラメタ r を有する $disengage$

e メッセージが到達した場合、

a) i はその親フィールドをクリアする。

【0078】b) i は範囲 $[rank(i) - 1, r]$ 外のランクを有するコンピュータを cri から削除し、新しい親を探し始め、すなわち、要求フェーズにはいる。

【0079】3. ack メッセージが代表の親から到達した場合、そのコンピュータは代表の親を真の親として記録する。

10 【0080】4. a) i はその代表の親のフィールドをクリアする。

【0081】b) ステップ2bを実行する。

【0082】図1はツリー世代の要求フェーズを示す流れ図であり、図2、図3、図4はツリー世代の受信フェーズを示す流れ図である。

【0083】ツリー世代の例として、ランク0からランク4を有し、かつ、2人の子を受け取ることができる5つのコンピュータを考察する。親は無作為に選択されるため、その方法により図5に示す段階に到達することができる。

【0084】コンピュータ1はコンピュータ0のみを親として選択することができるが、コンピュータ0は既に2人の子を有する。そのため、その方法は、コンピュータ0の子3とコンピュータ1を置換し、コンピュータ3にパラメタ1を有する $disengage$ メッセージを送信する。よって、図6に示す状態になる。

【0085】ノード3は範囲2から1までの親を探索し、例えば、ノード1を親として取り上げる。その結果得られるツリーは図7に示すようになる。

30 【0086】ノードが全て生きており、かつ、前の子がない場合、ツリー世代は1つのツリーで終了する。

【0087】上述したツリー世代は不平衡ツリーを構築するか、あるいは、1人の子のみを有する多くのノードを有するツリーを構築することができる。しかし、負荷平衡方法はツリーのトポロジーにより影響されるが、その影響は無視できる程度のものであることが分かっている。

【0088】ツリーの保守は、死んだノードを検知するか、新しいノードを負荷するか、あるいは、リブートされたノードに再コネクションするかするために必要である。ノードに障害が発生し、かつ、ノードの中には保守のために遮断されるものがあるため、ツリーの保守が必要である。本発明に係る実施例では、このような場合、次のようにして処理する。すなわち、

1. 各コンピュータは周期的に $update_up$ メッセージをその親に送信し、 $update_down$ メッセージを待つ。

【0089】2. 指定した期間内に応答がない場合、その親は死んだものと印が付けられ、そのコンピュータは新しい親を探す。

【0090】3. `update_up`メッセージが子から到達した場合、そのノードは`update_down`メッセージに応答する。

【0091】4. 指定した期間内に`update_up`メッセージが子から到達した場合、その子は「前の」と印が付けられる。

【0092】5. `update_up`メッセージが前の子から到達した場合、その子に対する前のフラグが送信され、そのノードは`update_down`メッセージに応答する。

【0093】6. コンピュータがリブートされるか、あるいはそのコンピュータに付加されると、そのコンピュータは親を探し、すなわち、要求フェーズに入る。

【0094】7. 各コンピュータは既に試みたコンピュータと死んだものと印が付けられたコンピュータとにフラグを立てる。これらのフラグは周期的にクリアされ、CRがリセットされ、より下位のランクを有する全てのコンピュータを含む。従って、リブートされたノードを再びプローブすることができる。

【0095】例として、図8に示すようにaからrのラベルを付けた18個のノードのツリーを考察する。

【0096】ノードhに障害が発生した場合、予め規定した期間が経過した後、その親cはそのノードhを「前の」と印をつけ、そのノードhが`update_up`メッセージを受信していないので、そのノードhを無視することになる。`update_down`メッセージが到達していないので、ノードkおよびrは新しい親を探し始める。今、そのネットワークは図9に示すように多くのばらばらのツリーを備える。

【0097】ノードkおよびrはそれらのFPの中に新しい親を探す。それらのFPが空である場合は、ノードkおよびrは無作為に親を探す。その新しいツリーは図10に示すツリーのようにすることができ、ノードkはノードgをその新しい親として持ち、ノードrはノードcをその新しい親として持つ。

【0098】ツリー保守機構と一緒になったツリー世代により、ネットワークのコンピュータが一般的に単一のツリー構造に配置されることになることが確認される。このことがノード障害の場合に真であることを知るため、ノードkに障害が発生するか、あるいはノードkが動作不能になっている任意のツリーについて考察してみよう。ノードkの親はノードkを「前の」と印を付け、ノードkの子はノードkを死んだものと印を付けることになる。ノードmがノードkの子だったと仮定しよう。そのため、ノードmは新しい親を探す。次の3つの可能性がある。

【0099】1. ノードmは新しい親を子として受け取るノードを見つけ出す。よって、単一のツリーが形成される。

【0100】2. ノードmはkの親を代表として取り出

すが、自由スロットはない。

【0101】この場合、「前の」と印が付けられたノードkのスロットが用いられ、ノードkは死んだものと印が付けられる。再び、単一のツリーが形成される。

【0102】3. ノードmは新しい親を子として受け取るノードを見つけ出さない。

【0103】このことは、新しい親を受け取るであろうノード（少なくとも1つ、すなわち、 $m-1$ ）が既に試みられていると印を付けられるか、あるいは、死んだものと印が付けられるかした場合にのみ起こる。この場合、そのネットワークは単一のツリーを備えるというよりは、一時的に数多くのサブツリーを備える。この状態は、全てのフラグがリセットされた（保守機構のステップ7）ときに終了し、ついで、ノードmは再び試み、かつ、親を見つけ出すことになる。

【0104】上述した理由付けはノードkの全ての子に対して適用可能である。

【0105】情報の交換と保守

実行待ち行列上のジョブの数は適正な負荷メトリックとして受け取られるのが一般的である。実行待ち行列が予め規定したしきい値、すなわち、`overload_mark`を超えた場合、ノードは過負荷と考えられる。実際に興味ある量はそのノードが`overload_mark`をどの程度超えるかということであるので、本発明に係る実施例では、僅かに異なるメトリックが用いられている。本発明に係る実施例では、負荷は`length_of_run_queue-overload_mark`として規定されている。例えば、`overload_markA=10`であり、かつ、`overload_marks=8`であり、かつ、ノードAおよびBの実行待ち行列の長さがそれぞれ5および4である場合は、`loadA=-5`であり、かつ、`loadB=-4`である。したがって、ノードbの負荷はノードAの負荷より大きい。

【0106】このメトリックが負荷に対して用いられる次の場合や、負荷という語が用いられる場合にはいつでも、`length_of_run_queue-overload_mark`の意味である。

【0107】ネットワークの各コンピュータは、そのネットワークの他のコンピュータ上の情報を保持する、分類されたベクトルを記憶する。そのベクトルの各項目は次のものを含む。

【0108】1) コンピュータのランク

2) コンピュータのローカル負荷

3) 負荷情報が伝送されるエッジまたはリンクを単位とした距離

4) 子の情報を伝播する最後のノード（最後のノードフィールド）

負荷ベクトルの長さ、すなわち、ベクトルが含む項目の数はノードからノードに変化させることができる。

【0109】負荷情報は次のようにしてネットワーク上を配送される。

【0110】1. 周期的に、各コンピュータはその負荷Lを抽出し、Lという新しい値を用いてその負荷ベクトルを分類し、負荷ベクトルをupdate_upメッセージに入れてその親に送信する。また、update_upメッセージはツリー保守のため動作可能であることを示すものとして用いられる。

【0111】2. ノードkからのupdate_upメッセージがノードmにより受信されると、

a. 受信された負荷ベクトルの各項目の長さは1だけインクリメントされる。

b. 受信するノードmから発信され、かつ、受信された負荷ベクトルの項目は全て削除される。

【0112】c. 送信するノードkから発信された負荷ベクトルの項目は全て削除される。

d. ローカルベクトルおよび受信されたベクトルの両方に現れる項目（コンピュータ）の距離は比較される。

【0113】1. ローカルベクトル項目の距離が受信されたベクトル項目の距離より大きい場合、あるいは等しい場合、ローカルベクトル項目が削除される。

【0114】2. 受信されたベクトル項目の距離がローカルベクトル項目の距離より大きい場合、受信されたベクトル項目が削除される。

【0115】e. 受信されたベクトルの項目の最後のノードフィールドは、送信するノードkになるようにセットされる。

【0116】f. 受信されたベクトルの項目はローカル負荷ベクトルと併合される。

【0117】3. 親はその新しい負荷ベクトルを、update_upメッセージが送信された子に、送信する。このメッセージはupdate_downメッセージであり、update_downメッセージは、その親が動作可能であることを示すものとしてツリー保守メッセージで再び用いられる。

【0118】4. update_downメッセージが到達すると、それを受信する子は受信された負荷ベクトルをステップ2で記載したのと同様の方法で処理する。

【0119】update_downメッセージが到達すると、メッセージを受信する子は受信された負荷ベクトルをステップ2と同様にして処理する。

【0120】update_downメッセージは、ツリー構造にリンクされている各ノードから、各ノードにより受信された情報を伝播する。よって、子はその親に*

$$rate2 = \max \{ effective_rate_n \} / \alpha$$

である。 α は次に示すコンピュータシミュレーション実験により決定される減衰係数である。

※

$$effective_rate_p = \max \{ rate1, rate2 \}$$

である。

【0128】コンピュータシミュレーション実験では、

*渡して情報を共用し、その親は情報を他の子に渡す。負荷情報をツリー内で上下に伝播することにより、別のサブツリーの標準より小さい負荷のノードに関する情報が任意のノードに到達することができる。負荷ベクトルのサイズが固定されているので、最低負荷状態のコンピュータの部分集合に関する情報を保持する。

【0121】負荷ベクトル内の負荷情報は、任意の特定のコンピュータの実際の負荷の近似のみである。というのは、その負荷情報が別のノードに到達するまで、負荷情報を発信するコンピュータの負荷は変化することができる。

【0122】「update間隔」は連続する負荷分散、すなわちupdate_upメッセージの時間間隔として規定されている。この時間間隔は負荷平衡の結果に対して大幅な影響を与える。値が非常に大きいため、他のノードの現在ノード負荷の情報が不正確になる。その情報が不正確なので、この情報に基づく移行判定結果は拒否が多くなる。値が非常に小さいので、すなわち、updateレートが大きいので、その方法のオーバーヘッドが増加し、かつ、応答時間が低下する。そのupdateレート（1/update間隔）はローカルプロセスの発生レートに正比例し、かつ、このレート変化のように変化すべきである。さらに、考慮に入れる要素が他にもある。現在のノードのupdateレートは、その子からその親、あるいはその親からその子に対する情報伝播が減速されないことが確認されなければならない。現在ノードのレートが小さく、かつ子または親のレートが大きい場合に、情報伝播が減速されることになる。

【0123】そのため、中間の隣接ノードのupdateレートをまた考慮しなければならない。

【0124】ノードのeffective_rateはこれら2つの要素に基づきレート計算関数の結果として規定されている。本発明の本実施例では、次の関数が用いられる。まず、各ノードのeffectiveレートはローカルプロセスの誕生レートにセットされる。各ノードに対して、次の量が計算される。

【0125】

a) $rate1 = local_birth_rate_p$
ローカル誕生レートpはスライディングタイムウィンドウに亘る平均として計算される。

【0126】b) pの全ての中間隣接ノードnに対して、

※【0127】pのeffective_rateは

1. 4から2.0の範囲内の α の値は、負荷が均等に分散されたネットワークに対して良い結果が得られ、大き

い負荷の領域を有するネットワークに対して良い結果が得られる。

【0129】コンピュータが過負荷になると、余分の負荷を受け取ることができるコンピュータを示す第1項目に対するローカル負荷ベクトルを探索し、余分の負荷を受け取ることができるコンピュータに、1つ以上のタスクを直接転送する。負荷ベクトル内の目標コンピュータの負荷項目は更新され、負荷ベクトルは再分類される。目標コンピュータが何等かの理由で余分の負荷を受け取ることができない場合は、送信元が通知される。この場合、送信元は負荷ベクトルからその項目を削除し、次の項目を検索し、検索された項目を新しい目標として用いる。余分の負荷を受け取ることができるノードが1つもない場合は、移行は開始されない。

【0130】したがって、位置指定方策、すなわち、目標ノードを見付け出すことが、負荷ベクトルを分類するのに用いられる分類基準に依存する。負荷情報はホップしてツリー上を伝播されるので、ノードはさらに遠くなり、情報の精度が落ちる。そのため、この評価例では、負荷ベクトルは負荷および距離に基づき分類される。項目はそれらの負荷に応じてまず分類される。そして、等しい負荷を有する項目は距離の降順で分類される。

【0131】長い時間、軽負荷であるノードは、多くのノードの負荷ベクトル内の第1項目として現れる。このため、多くのノードはプロセスをそのノードに移行させて、そのノードを溢れさせる。このことがボトルネックとなる。この問題は2つのアプローチを用いて避けることができる。

【0132】等価ノードの集合を、負荷ベクトル内で、同一負荷および距離を有するノードとして規定することができる。例えば、負荷1および距離2を有するノードは全て同一の等価集合内にある。この場合、分類基準により、同一等価集合内のノードは全てベクトルで群分けされることが確認される。負荷ベクトルが(update_upまたはupdate_downメッセージにより)分散されるまで、ベクトル内の第1等価集合が置換される。このため、同一ノードが多くの負荷ベクトルに現れる確率が小さくなる。

【0133】さらに、まず、軽い負荷のノードがそれ自体の負荷ベクトルに最初に現れるといつも、カウンタが*

a b c d

第1クラス

次のステップを実行してそれらのノードをランク付けする。

a b c d

1 2 0 3

2. 各クラスを順番に取り上げ、前のノードの数だけ、各ノードのランクを増やす。例えば、第2クラス

a b c d

8 9 7 10

*項目に接続され、しかも、初期化され、常に零未満の軽負荷ノードの負荷に等しくされる。ベクトルが分散されるといっても、その項目がまだ第1である場合、カウンタがインクリメントされる。カウンタが正になる場合、その項目がそのベクトルから削除される。よって、軽負荷のノードを知るノードの数が制限され、ノード溢れの確率が小さくなる。

【0134】プロセス移行の開始は、ローカル負荷が検査される周波数に依存するとともに、ローカル負荷の値に依存する。ノード負荷ステータスを検査するか否かの判定は、選択された方策に依存する。可能な方策は数多くあり、次のようなものを含む。

【0135】1. 周期的 ローカル負荷はupdate_upメッセージを送信した後、試験される。すなわち、ローカル負荷抽出はupdate_upメッセージに同期される。

【0136】2. ドライブされる事象 ローカル負荷はローカルプロセスが実行待ち行列に付加されるときはいつでも試験される。

【0137】コンピュータの過負荷をいつ宣言するかかの判定、すなわち、過負荷マークをどこに置くかの判定は、その方法の実施に影響を与える。過負荷マークを余り低く置くと、移行が余りにも多くなり、応答時間が改善されない。過負荷マークを余り高く置くと、応答時間が低下する。過負荷マークの値は任意の特定のシステムに対して最適である必要があり、その値は、異なる負荷容量を有するネットワーク内のコンピュータに対するか、あるいは異なるジョブプロファイルに対して、異ならせることができる。最適な1つの値または複数の値を、所定の状態で、例えば、適正なコンピュータシミュレーション実験を利用するか、あるいは、試行錯誤のプロセスにより、確立することができる。

【0138】各ノードの構成ファイル内にランクを適正に割り当てて、ノードの制御されたクラス化を行うことができる。

【0139】例えば、aからkの11個のコンピュータがある場合、次のように、4つのコンピュータのクラスと7つのコンピュータのクラスを形成する必要がある。

e f g h i j k

第2クラス

※【0140】1. 各クラスは別々にランク付けされる。例えば、

e f g h i j k

5 6 0 1 2 3 4

をまず選択した場合、

e f g h i j k

5 6 0 1 2 3 4

21

異なるランク付けファイルをコンピュータに割り当てることにより、クラスタが構成される。クラスタ内で最低ランクを有するコンピュータを除き、各コンピュータは、そのクラスタ内のノードのランクを含むが、そのクラスタ外のノードのランクを含まないランク付けリストを記憶する。その最低のランクを有するコンピュータは他のクラスタのノードを有するランク付けリストを記憶する。

【0141】そのため、クラスタのノードは他のノードを知らないで、相互接続されることになる。最低のランクを有するノードのみが別のクラスタのノードに接続されることになる。図11は上記例のノードから形成されたサンプルツリーを示す。

【0142】cに対するこの最高数が3である場合、a, b, およびdの好ましい親リストはcのみを含み、その結果、図12に示すツリーが得られる。

【0143】cの好ましい親リスト内にjを置くと、図13に示すツリーが生成される。

【0144】本発明の実施例では、各ノードに対するランク付けファイルと好ましい親リストを賢明に選択することにより、クラスタ化が行われる。

【0145】以上、過負荷ノードによりタスクの転送が開始される送信元開始の負荷平衡方策を用い、本発明の実施例を説明したが、本発明は、受信先開始方策、すなわち、軽負荷のコンピュータが負荷の大きいコンピュータから自分自身にタスクの転送を開始する方策に、同様に適用可能である。

【0146】ネットワーク内のコンピュータを操作する方法と、改善された負荷平衡方策を用いてコンピュータのネットワークを操作する方法とを記載してきた。この改善された負荷平衡方策は拡張可能であり、フォールトトレラントであり、柔軟性があり、しかも、クラスタ化をサポートしている。よって、クラスタ化を、コンピュ

22

ータの数が少ないネットワークからその数が非常に多いネットワークまでの利用に適合させている。

【0147】

【発明の効果】以上説明したように、本発明によれば、上記のように構成したので、より速く負荷を平衡させることができるという効果がある。

【図面の簡単な説明】

【図1】 ツリー生成の要求および受信フェーズを示す流れ図である。

10 【図2】 ツリー生成の要求および受信フェーズを示す流れ図である。

【図3】 ツリー生成の要求および受信フェーズを示す流れ図である。

【図4】 ツリー生成の要求および受信フェーズを示す流れ図である。

【図5】 本発明に係るツリー生成の段階を示す図である。

【図6】 本発明に係るツリー生成の段階を示す図である。

20 【図7】 本発明に係るツリー生成の段階を示す図である。

【図8】 本発明に係る実施例におけるツリー保守を説明するための説明図である。

【図9】 本発明に係る実施例におけるツリー保守を説明するための説明図である。

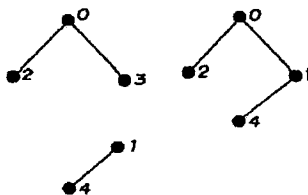
【図10】 本発明に係る実施例におけるツリー保守を説明するための説明図である。

【図11】 本発明に係る実施例におけるノードのクラスタ化処理を説明するための説明図である。

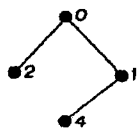
30 【図12】 本発明に係る実施例におけるノードのクラスタ化処理を説明するための説明図である。

【図13】 本発明に係る実施例におけるノードのクラスタ化処理を説明するための説明図である。

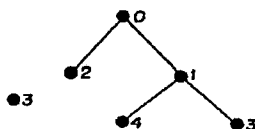
【図5】



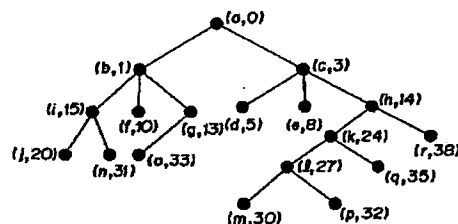
【図6】



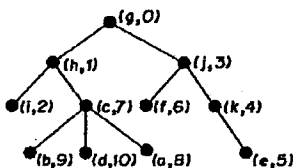
【図7】



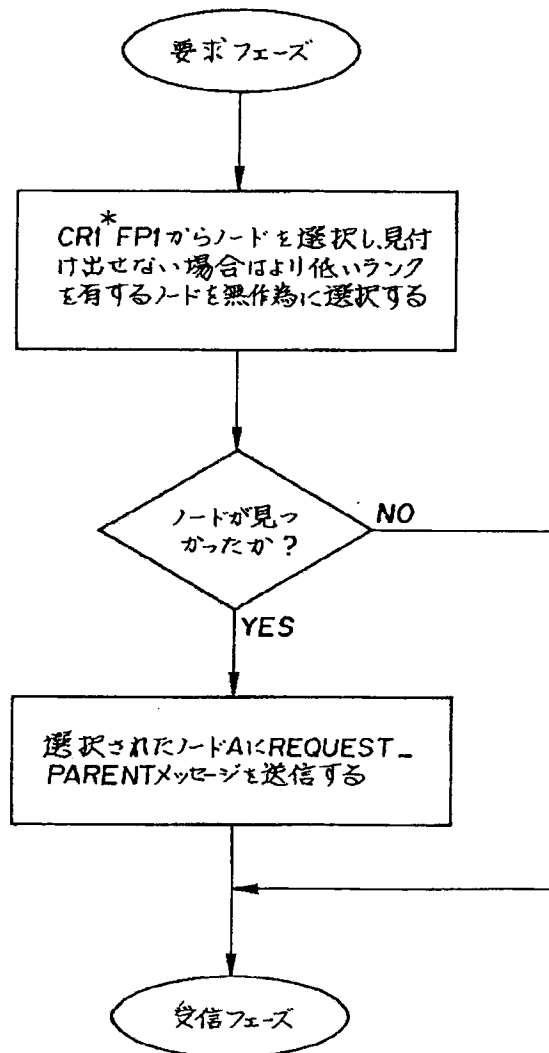
【図8】



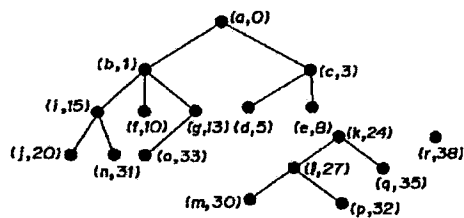
【図12】



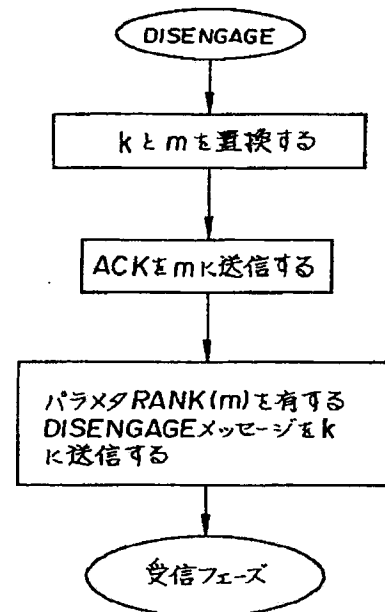
【図1】



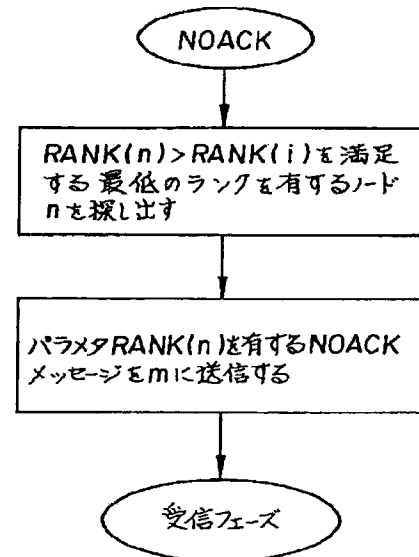
【図9】



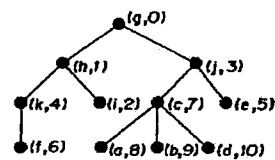
【図3】



【図4】



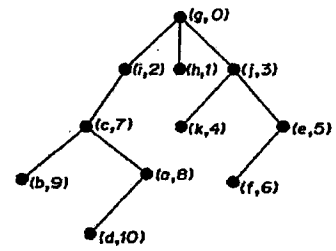
【図13】



```

graph TD
    Start([受信フェーズ]) --> Read[ノードmからメッセージを読み取る]
    Read --> Ack{ACK?}
    Ack -- YES --> Verify[mを真の親として印を付け、  
論理リンクを確立する]
    Verify --> Read
    Ack -- NO --> ParentReq{親要求?}
    ParentReq -- YES --> FreeSlot{自由スロット?}
    FreeSlot -- YES --> DelPrev[前の子  
削除]
    DelPrev --> SendAck[ACKをm  
に送信]
    SendAck --> MakeLink[mへのリンク  
を作成]
    MakeLink --> End1([受信フェーズ])
    FreeSlot -- NO --> PrevChild{前の子?}
    PrevChild -- YES --> DelPrev
    PrevChild -- NO --> RankCheck{RANK(k) > RANK(m)?}
    RankCheck -- YES --> Disengage1([DISENGAGE])
    RankCheck -- NO --> NoAck([NOACK])
    NoAck --> ParentReq
    ParentReq -- NO --> Disengage2{DISENGAGE(r)?}
    Disengage2 -- YES --> ClearParent[親ファールドをクリア]
    ClearParent --> End2([要求フェーズ])
    Disengage2 -- NO --> NoAckR{NOACK(r)?}
    NoAckR -- YES --> ClearParent
    NoAckR -- NO --> PrintErr[印刷誤り  
メッセージ]
    PrintErr --> End3([受信フェーズ])
  
```


【図 1 1】



(72) 発明者 ヨセフ モアティ
イスラエル ハイファ ハニタ ストリー
ト 68/56

(72) 発明者 アブラハム テパーマン
イスラエル ハイファ ハビバ ライヒ
ストリート 46

* NOTICES *

JP0 and INPIT are not responsible for any damages caused by the use of this translation.

- 1.This document has been translated by computer. So the translation may not reflect the original precisely.
- 2.**** shows the word which can not be translated.
- 3.In the drawings, any words are not translated.

EXAMPLE

[Example]The example of this invention is described with reference to drawings.

[0055]The example concerning this invention consists of the two principal parts. The 1st portion is tree construction and maintenance. The 2nd portion is exchange and maintenance of load-balancing information.

[0056]The rank of a meaning used for tree construction is assigned to each computer on tree construction and a maintenance network. Hereafter, in order that the word of "parents" may refer to the computer of a low rank rank, it is used, and one computer can form the link of a higher rank to the computer of this low rank. The word of a "child" can say the computer of a higher rank rank, and one computer can form a low-ranking link to the computer of this higher rank rank.

[0057]Each computer memorizes the configuration file containing following (i) - (v), and accesses the configuration file.

[0058](i) The maximum number of the posterity of the direct system which the computer can receive. These numbers are called child slot.

[0059](ii) The list of computers, and each rank of a computer.

[0060]" (iii) -- the ordering list FP of friendly parent" nodes. Friendly parents' rank is a low rank from a computer now. FP list is used by computer and chooses a parent node. FP list is chosen arbitrarily. FP list can be emptied when parents are chosen at random. FP list contains the subset of all the low-ranking computers.

[0061](iv) Time to wait for a response of parents and a child.

[0062](v) Time to stand by in order to carry out the probe of whether the node is dead or to be alive.

[0063]Although the list of a candidate's parents to computer CR is called candidate range of P, it limits the range of possible parents. At first, CR of each computer contains all the computers of a rank lower than the computer. Only when a node is in CR and FP, the node is chosen as parents. Only the member of CR is inspected, when FP is empty or it uses up.

[0064]Each computer carries out the following step in order to build a tree. There is a demand phase, one computer chooses another computer in this phase, and it demands to become parents. There is a receiving phase and a computer answers the demand from waiting and other computers in a response in this phase.

[0065]Demand phase 1. each computer i takes out another computer from friendly parents' list FP_i . When

FP_i goes wrong in all the candidates of empty or FP_i , a low-ranking computer is chosen from it at random. When there is no selected computer j in CR_i , the next computer of FP_i is chosen or another computer is taken out at random. When the trial of all the nodes goes wrong, the computer enters at a receiving phase. [0066]2. i transmits a request_parent message to the selected computer j , and waits for the message from other computers.

[0067]3. When the parents of representation do not answer within a certain period, the parents of this representation can put the dead thing and a seal, and i enters again at a demand phase.

[0068]When a demand phase 1.request_parent message reaches from m and (a) i has a free child slot, i receives the demand of m and transmits an ack message.

[0069](b) When a number of child slots with which i was assigned are used up, $1i$ looks for the child before scanning a child and specifying below. the case where the former child is found -- (a) -- the child can put the dead thing and a seal and is deleted.

[0070](b) Make a child slot available.

[0071](c) i transmits an ack message to m .

[0072]When the former child is not found, $2i$ finds the child k who has the highest rank.

[0073]3) When it is $\text{rank}(k) > \text{rank}(m)$, ai accepts m as a child instead of k , and transmits an ack message to m .

[0074]b) i transmits the disengage message which contains $\text{rank}(m)$ in k as a parameter.

[0075]4) When it is $\text{rank}(k) < \text{rank}(m)$, ai places the computer n which has the lowest rank that is satisfied with the list of [in a computer] of $\text{rank}(k) > \text{rank}(m)$.

[0076]b) i transmits the noack message which contains $\text{rank}(n)$ in m as a parameter.

[0077]2. When the disengage message which has the parameter r reaches, ai clears the parent field.

[0078]b) i deletes the computer which has a rank besides the range $[\text{rank}(i)-l, r]$ from cr_i , and begins to look for new parents, namely, is at a demand phase.

[0079]3. When an ack message reaches from the parents of representation, the computer records the parents of representation as true parents.

[0080]4.a) i clears the field of the parents of the representation.

[0081]b) Perform step 2b.

[0082]Drawing 1 is a flow chart showing a tree generation's demand phase, and drawing 2, drawing 3, and drawing 4 are the flow charts showing a tree generation's receiving phase.

[0083]Five computers which have the rank 4 from the rank 0, and can receive two children as a tree generation's example are considered. Since parents are chosen at random, they can reach the stage shown in drawing 5 by the method.

[0084]Although the computer 1 can choose only the computer 0 as parents, the computer 0 already has two children. Therefore, the method replaces the child 3 of the computer 0, and the computer 1, and transmits the disengage message which has the parameter 1 to the computer 3. Therefore, it will be in the state which shows in drawing 6.

[0085]The node 3 looks for the parents to the ranges 2-1, for example, takes up the node 1 as parents. The tree obtained as a result comes to be shown in drawing 7.

[0086]When all nodes are valid and there is no former child, a tree generation is ended by one tree.

[0087]The tree generation who mentioned above can build the tree which has many nodes which build a disequilibrium tree or have only one child. However, although the load-balancing method is influenced by the topology of a tree, it turns out that the influence is a thing of the grade which can be disregarded.

[0088]Maintenance of a tree is required in order [which carries out a re-connection to the node which detected the dead node, carried out load of the new node, or was rebooted] to graze. Since there are some which an obstacle occurs in a node and are intercepted in a node for maintenance, maintenance of a tree is required. In such a case, in the example concerning this invention, it processes as follows. That is, 1. each computer transmits a update_up message to the parents periodically, and waits for a update_down message.

[0089]2. When there is no response within the specified period, that from which the parents died, and a seal are put, and the computer looks for new parents.

[0090]3. When a update_up message reaches from a child, the node answers a update_down message.

[0091]4. When a update_up message reaches from a child within the specified period, as for the child, a seal is put a "front."

[0092]5. When a update_up message reaches from the former child, the flag before receiving the child is transmitted and the node answers a update_down message.

[0093]6. If a computer is rebooted or it is added to the computer, the computer will look for parents, namely, will enter at a demand phase.

[0094]7. Each computer sets a flag to the already tried computer, the dead thing, and the computer by which the seal was put. It is cleared periodically, CR is reset and these flags contain all the computers which have a low-ranking rank more. Therefore, the probe of the rebooted node can be carried out again.

[0095]The tree of 18 nodes which attached the label of a to r as an example as shown in drawing 8 is considered.

[0096]Since the parents c put a seal for the node h a "front" and the node h has not received the update_up message after the period specified beforehand passes when an obstacle occurs in the node h, the node h will be disregarded. Since the update_down message has not reached, the nodes k and r begin to look for new parents. Now, the network is provided with many scattering trees as shown in drawing 9.

[0097]The nodes k and r look for new parents in those FP(s). When those FP(s) are empty, the nodes k and r look for parents at random. The new tree can be carried out like the tree shown in drawing 10, the node k has the node g as the new parents, and the node r has the node c as the new parents.

[0098]It is checked by the tree generation who became together with a tree maintenance mechanism that a network computer will generally be arranged at a single tree structure. When this is a node failure, in order to know that it is truth, I will consider the arbitrary trees in which an obstacle occurs in the node k, or the node k has become impossible of operation. The parents of the node k will put a seal for the node k a "front", and the child of the node k will put what died the node k, and a seal. I will assume that the node m was a child of the node k. Therefore, the node m looks for new parents. There are the following three possibilities.

[0099]1. The node m finds out the node which receives new parents as a child. Therefore, a single tree is formed.

[0100]2. Although the node m takes out the parents of k as a representative, it does not have a free slot.

[0101]In this case, the slot of the node k on which the seal was put the "front" is used, and that from which the node k died, and a seal are put. Again, a single tree is formed.

[0102]3. The node m does not find out the node which receives new parents as a child.

[0103]This happens, only when [which was lent] what could put the seal or died, and a seal will be put, if the node (at least one, i.e., m-1) which will receive new parents is already tried. In this case, that network is temporarily provided with many subtrees rather than having a single tree. This state will be ended when all the flags are reset (Step 7 of a maintenance mechanism), subsequently the node m will be tried again, and parents will be found out.

[0104]Reason attachment mentioned above is applicable to all the children of the node k.

[0105]As for the number of exchange of information, and the jobs on maintenance execution queuing, being received as proper load metric is common. A node is considered to be an overload when execution queuing exceeds the threshold specified beforehand, i.e., overload_mark. Metric one from which it differs [node / the] slightly in the example concerning this invention since an actually interesting quantity is how many overload_mark to exceed is used. Load is prescribed as length_of_run_queue-overload_mark by the example concerning this invention. For example, it is load_A=-5, when it is overload_mark_A=10, it is overload_mark_B=8 and the length of execution queuing of the nodes A and B is 5 and 4, respectively.

And it is load_B=-4.

Therefore, the load of the node b is larger than the load of the node A.

[0106]The case of the next for which metric one of this is used to load, and when the word of load is used, it is a meaning of length_of_run_queue-overload_mark always.

[0107]Each network computer memorizes the classified vector holding the information on the computer of everything but the network. Each item of the vector contains the following.

[0108]1) The node of the last which spreads the information of distance 4 child who made the unit the edge or link with which local load 3 load information of rank 2 computer of a computer is transmitted (the last node field)

The length of a load vector, i.e., the number of the items which a vector includes, can be changed from a node to a node.

[0109]Load information has a network top delivered as follows.

[0110]1. Periodically, each computer extracts the load L, classifies the load vector using the new value L, puts a load vector into a update_up message, and transmits to the parents. A update_up message is used as what shows that it can operate for tree maintenance.

[0111]2. The node's m reception of the update_up message from the node k will ***** the length of each item of the load vector of which a. reception was done only 1.

b. All the items of the load vector which was sent from the node m which receives and was received are deleted.

[0112]c. All the items of the load vector sent from the node k which transmits are deleted.

d. The distance of the item (computer) which appears in both a local vector and the received vector is compared.

[0113]1. It is larger than the distance of the vector item by which the distance of the local vector item was

received, or when equal, a local vector item is deleted.

[0114]2. When the distance of the received vector item is larger than the distance of a local vector item, the received vector item is deleted.

[0115]e. The node field of the last of the item of the received vector is set so that it may become the node k which transmits.

[0116]f. The item of the received vector is merged into a local load vector.

[0117]3. Parents transmit the new load vector to the child by whom it was transmitted to the update_up message. This message is a update_down message and a update_down message is again used by a tree maintenance message as what shows that those parents can operate.

[0118]4. If a update_down message reaches, the child who receives it will process the received load vector by the same method as Step 2 indicated.

[0119]If a update_down message reaches, the child who receives a message will process the received load vector like Step 2.

[0120]A update_down message spreads the information received by each node from each node linked to the tree structure. Therefore, a child hands the parents, information is shared and the parents hand other children information. By spreading load information up and down within a tree, nodes with arbitrary information about the node of load smaller than the standard of another subtree can be reached. Since the size of the load vector is being fixed, the information about the subset of the computer of the minimum loaded condition is held.

[0121]The load information within a load vector is only approximation of the actual load of arbitrary specific computers. Because, the load of the computer which sends load information can change until the load information reaches another node.

[0122]The "update interval" is specified as a time interval of continuous load sharing, i.e., a update_up message. This time interval has large influence to the result of load balancing. Since the value is very large, the information on the present node load of other nodes becomes inaccurate. Since that information is inaccurate, the refusal of the shift decision result based on this information increases. Since the value is dramatically small (i.e., since the update rate is large), the overhead of the method increases and response time falls. That update rate ($1/\text{update interval}$) should be in direct proportion to the occurrence rate of a local process, and should change like this rate change. There are other elements taken into consideration. As for the update rate of the present node, it must be checked that the information propagation to the parents or its child from the parents is not slowed down from the child. Small [the rate of the present node], when the rate of a child or parents is large, information propagation will be slowed down.

[0123]Therefore, the update rate of a middle adjacent node must be taken into consideration again.

[0124]effective_rate of the node is specified as a result of a rate calculation function based on these two elements. The following function is used in this example of this invention. First, the effective rate of each node is set to the birth rate of a local process. The following quantity is calculated to each node.

[0125]

a) The $\text{rate}_1 = \text{local_birth_rate}_p$ local birth rate p is calculated as an average covering a slide time window.

[0126]b) It is $\text{rate}_2 = \max\{\text{effective_rate}_n\} / \alpha$ to all the middle adjacent node n of p. alpha is a damping

coefficient determined by the computer simulation experiment shown below.

[0127]effective_rate of p is $\text{effective_rate}_p = \max \{ \text{rate}_1, \text{rate}_2 \}$.

It comes out.

[0128]In a computer simulation experiment, a good result is obtained the network [for] with which a good result is obtained the network [for] with which load was distributed uniformly, and the value of alpha of 1.4-2.0 within the limits has a field of large load.

[0129]If a computer becomes an overload, it will search for the local load vector over the 1st item that shows the computer which can receive excessive load, and transfer direct of the one or more tasks will be carried out to the computer which can receive excessive load. The load item of the target computer within a load vector is updated, and reclassification of the load vector is carried out. A transmitting agency is notified when a target computer cannot receive excessive load for a certain reason. In this case, a transmitting agency deletes that item from a load vector, searches the following item and uses the searched item as a new target. Shift is not started when one does not have a node which can receive excessive load.

[0130]Therefore, it is dependent on the standard of classification used for classifying a load vector to find out a tab-control-specification policy, i.e., a target node. Since load information hops and a tree top is spread, a node becomes still further and the accuracy of information falls. Therefore, a load vector is classified according to this example of evaluation based on load and distance. An item is first classified according to those loads. And the item which has equal load is classified according to the descending order of distance.

[0131]The node which are long time and a light load appears as the 1st item within the load vector of many nodes. For this reason, many nodes make a process shift to that node, and flood that node. This serves as a bottleneck. This problem is avoidable using two approaches.

[0132]A set of an equivalent node can be specified within a load vector as a node which has the same load and distance. For example, all the nodes that have load-1 and the distance 2 are in the same equivalent set. In this case, it is checked by a standard of classification that the group division of all the nodes in the same equivalent set is carried out by a vector. The 1st equivalence set within a vector is replaced until a load vector is distributed (update_up or update_down message). For this reason, the probability that the same node will appear in many load vectors becomes small.

[0133]Whenever the node of light load appears first in the load vector of itself, it is connected to an item, and moreover, a counter is initialized and is always made equal to the load of the light load node of less than zero. Whenever a vector is distributed, it *****s a counter, when the item is still the 1st. When a counter just becomes, the item is deleted from the vector. Therefore, the number of the nodes which get to know the node of a light load is restricted, and the probability of node overflow becomes small.

[0134]It depends for the start of process shift on the value of local load while it is dependent on the frequency on which local load is inspected. The judgment of whether to inspect node load status is dependent on the selected policy. There are many possible policies and they contain the following.

[0135]1. Periodic After local load transmits a update_up message, it is examined. That is, local load extraction synchronizes with a update_up message.

[0136]2. Phenomenon driven Local load is examined always, when a local process is added to execution queuing.

[0137]The judgment of when to declare the overload of a computer, i.e., the judgment of where to place an overload mark, affects operation of the method. If an overload mark is placed not much low, shift will increase too much and response time will not be improved. If an overload mark is placed not much highly, response time will fall. It can be made to differ to a job profile which is [as opposed to / the computer in the network which has the load carrying capacity from which the value of an overload mark needs to be the optimal to arbitrary specific systems, and the value differs] different. About one optimal value or two or more values, it is in a predetermined state, and for example, a proper computer simulation experiment can be used, or it can be established by the process of trial and error.

[0138]A rank can be properly assigned in the configuration file of each node, and clustering by which the node was controlled can be performed.

[0139]For example, when there are 11 computers of a to k, it is necessary to form the cluster of four computers, and the cluster of seven computers as follows.

a b c d e f g h i j k The 1st cluster The following step is performed the 2nd cluster and those nodes are ranked.

[0140]1. Each cluster is ranked independently. For example, a b c d e f g h i j k 1 2 0 3 5 6 0 1 2 3 42. each cluster is taken up in order, and only the number of front nodes increases the rank of each node. For example, when the 2nd cluster is chosen first, it is a b c d e f g h i j k 8. 9 7 10 5 6 0 1 2 3 A cluster is constituted by assigning a rank file different four to a computer. Although each computer includes the rank of the node in the cluster except for the computer which has the minimum rank within a cluster, the rank list which does not include the rank of the node besides the cluster is memorized. The computer which has the minimum rank memorizes the rank list which has a node of other clusters.

[0141]Therefore, since the node of a cluster does not know other nodes, interconnection will be carried out. Only the node which has the minimum rank will be connected to the node of another cluster. Drawing 11 shows the sample tree formed from the node of the above-mentioned example.

[0142]When this highest number to c is 3, the tree which shows drawing 12 the desirable parent list of a, b, and d including [as a result] only c is obtained.

[0143]If j is placed into the desirable parent list of c, the tree shown in drawing 13 will be generated.

[0144]In the example of this invention, clustering is performed by choosing wisely the rank file to each node, and a desirable parent list.

[0145]As mentioned above, although the example of this invention was described using the load-balancing policy of the transmitting agency start by which transmission of a task is started by the overload node, A reception destination start policy, i.e., the computer of a light load, can apply this invention to the policy which starts transmission of a task for itself from a computer with large load similarly.

[0146]How to operate the computer in a network, and the method of operating the network of a computer using the improved load-balancing policy have been indicated. This improved load-balancing policy is extensible, fault-tolerant, is supple, and, moreover, is supporting clustering. Therefore, clustering is fitted to use from a network with few computers to a network with very many the number.

[Translation done.]